AMENDMENTS TO THE CLAIMS

Kindly replace the claims as follows.

1. (currently amended) A computer, comprising:

a general register file of registers;

a RISC instruction decoder exposed for execution of user-state programs in a RISC instruction set, being an instruction set having fixed-length instructions and a load/store/operate organization; and

a hardware CISC instruction decoder exposed for execution by user-state programs in a CISC instruction set, being an instruction set with variable-length instructions and many instructions having multiple side-effects, the CISC decoder designed to decode less than the entire CISC ~~a portion of an~~ instruction set for the computer, and to deliver the decoded instructions to an instruction execution pipeline designed to execute the output of both the RISC instruction decoder and the CISC instruction decoder;

a software emulator programmed to implement a remainder of the instruction set;

the CISC instruction set providing accessibility to only a subset of the registers of the general register file, intermediate results of instructions of the instruction set being stored in registers of the general register file that are inaccessible in the CISC instruction set;

a RISC instruction set of the RISC decoder designed to share a substantial portion of its opcode values with opcodes of the CISC instruction set, and to implement a substantial portion of the addressing modes of the CISC instruction set.

2. (currently amended) A method, comprising the steps of:

decoding instructions of a user-state program coded in a RISC instruction set in a hardware instruction decoder of a computer, the RISC instruction set being an instruction set having fixed-length instructions and a load/store/operate organization; and

decoding instructions of a user-state program coded in a CISC instruction set in a CISC hardware instruction decoder of a computer, the CISC instruction set being an

instruction set having variable-length instructions and many instructions having multiple side-effects; and

the instructions decoded by the CISC decoder and RISC decoder being executed in a common execution pipeline;

a RISC instruction set of the RISC decoder designed to share a substantial portion of its opcode values with corresponding opcodes of the CISC instruction set.

3. (original) The method of claim 2, wherein:

the pipeline exception circuitry is designed to recognize an exception occurring in a CISC instruction after a first side-effect of the CISC instruction has been architecturally committed, and thereupon, to expose an instruction pointer, contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer exposing sufficient processor state and providing sufficient working storage for execution of the exception handler and resumption of the program, without recomputing the first side-effect, without storing processor state to the main memory.

4. (original) The method of claim 2, wherein the computer further comprises:

a register and control logic for that register designed to capture and expose an intra-instruction program counter value when a CISC instruction raises an exception at an intermediate point.

5. The method of claim 2, wherein:

a first operating system is loaded into memory, being coded in the RISC instruction set;

a second operating system is loaded into memory, being coded in the CISC instruction set, the CISC operating system being unmodified for execution on the computer of the RISC instruction set;

hardware and/or software of the computer are designed to accept an exception occurring during execution of a program coded in the RISC instruction set, and to route the exception for handling in the CISC operating system; and

hardware and/or software of the computer are designed to accept an exception occurring during execution of a program coded in the CISC instruction set, and to route the exception for handling in the RISC operating system.

6. (original) The method of claim 2, further comprising the steps of:

during execution of a CISC instruction, in response to an operation of the instruction calling for an architecturally-visible side-effect in a storage location architecturally-visible in the CISC instruction set, storing a value representative of an architecturally-visible representation of the side-effect, a format of the representative value being different than an architecturally-visible representation of the side-effect, and resuming the execution without generating the architecturally-visible side-effect;

later writing the architecturally-visible representation corresponding to the representative value into the architecturally-visible storage location.

7. (original) The method of claim 2, further comprising the steps of:

during execution of a program in the CISC instruction decoder, recognizing a condition in which an instruction is to affect the execution of the following instruction, and in response, setting the processor into single-step mode;

taking a single-step exception after executing the following instruction, and setting the processor out of single-step mode.

8. (original) The method of claim 2, further comprising the steps of:

as a result of execution of a program in the CISC instruction decoder, the instruction opcode calling for a memory reference to effect a movement of data, performing a memory

protection check effective to check for permission to effect a movement of data other than the data movement called for by the instruction opcode.

9. (currently amended) A computer, comprising:

a RISC instruction decoder exposed for execution of user-state programs in a RISC instruction set, being an instruction set having fixed-length instructions and a load/store/operate organization; and

a CISC instruction decoder exposed for execution by user-state programs in a CISC instruction set, being an instruction set with variable-length instructions and many instructions having multiple side-effects;

an instruction execution pipeline designed to execute the output of both the RISC instruction decoder and the CISC instruction decoder;

a RISC instruction set of the RISC decoder designed to share a substantial portion of its opcode values with corresponding opcodes of the CISC instruction set.

10. (currently amended) The computer of claim 9, wherein:

the CISC instruction decoder implements ~~only~~ a portion of the CISC instruction set less than the entire CISC instruction set, a remainder of the CISC instruction set being implemented in a software emulator coded in the RISC instruction set.

11. (original) The computer of claim 9, wherein:

the CISC instruction set provides accessibility to only a subset of a general register file, intermediate results of instructions of the CISC instruction set being stored in registers of the general register file that are inaccessible in the CISC instruction set.

12. (original) The computer of claim 11, further comprising:

an exception handler for initiation by an exception occurring at an intermediate point during execution of one of the instructions of the first instruction set, the exception handler

being coded in the RISC instruction set having accessibility to the registers inaccessible in the CISC instruction set, any saving of the intermediate results as part of a save of machine state using mechanisms used for saving general registers.

13. (original) The computer of claim 9, wherein:

the pipeline exception circuitry is designed to recognize an exception occurring in a CISC instruction after a first side-effect of the CISC instruction has been architecturally committed, and thereupon, to expose an instruction pointer, contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer exposing sufficient processor state and providing sufficient working storage for execution of the exception handler and resumption of the program, without recomputing the first side-effect, without storing processor state to the main memory.

14. (original) The computer of claim 9, further comprising:

a register and control logic for that register designed to capture and expose an intra-instruction program counter value when a CISC instruction raises an exception at an intermediate point.

15. (original) The method of claim 9, further comprising:

a first operating system is loaded into memory, being coded in the RISC instruction set;

a second operating system is loaded into memory, being coded in the CISC instruction set, the CISC operating system being unmodified for execution on the computer of the RISC instruction set;

hardware and/or software of the computer are designed to accept an exception occurring during execution of a program coded in the RISC instruction set, and to route the exception for handling in the CISC operating system; and

hardware and/or software of the computer are designed to accept an exception occurring during execution of a program coded in the CISC instruction set, and to route the exception for handling in the RISC operating system.

16. (original) The computer of claim 9, further comprising:

circuitry designed to recognize an operation of an instruction calling for an architecturally-visible side-effect in an architecturally-visible storage location, and in response, to store a value representative of an architecturally-visible representation of the side-effect, a format of the representative value being different than an architecturally-visible representation of the side-effect, and to resume the execution without generating the architecturally-visible side-effect;

circuitry and/or software designed to later write the architecturally-visible representation corresponding to the representative value into the architecturally-visible storage location.

17. (original) The computer of claim 9, further comprising:

hardware and/or software designed to recognize a condition in which a CISC instruction is to affect the execution of the following instruction, and in response, to set the processor into single-step mode;

hardware and/or software designed to raise a single-step exception after executing the following instruction, and to set the processor out of single-step mode.

18. (original) The computer of claim 9, further comprising:

circuitry effective during execution of an instruction whose opcode calls for a memory reference to effect a movement of data, to perform a memory protection check effective to check for permission to effect a movement of data other than the data movement called for by the instruction opcode.

19. (original) The computer of claim 9, wherein the CISC instruction decoder generates instructions in the RISC instruction set for execution by the instruction execution pipeline.

20. (original) The computer of claim 19, wherein a last of the RISC instructions generated for each CISC instruction carries a marker indicating that it is the last RISC instruction for the CISC instruction.

21. (original) The computer of claim 19, wherein a plurality of the RISC instructions generated for a single CISC instruction carry a marker indicating that the computer may accept an exception at the marked RISC instruction.

22. (original) The computer of claim 19, wherein the CISC instruction decoder is designed to generate multiple RISC instructions for parallel execution.

23. (original) The computer of claim 19, further comprising hardware and/or software designed to accept multiple exceptions raised by the RISC instructions generated for a single CISC instruction and to collect the multiple exceptions for collected presentation to a CISC processing environment.

24. (original) The computer of claim 19, wherein the CISC instruction decoder and instruction execution pipeline, with at most limited exceptions, are designed to independently complete the RISC instructions generated for CISC instructions once the CISC instructions are issued to the instruction execution pipeline.

25. (original) The computer of claim 19, wherein the instruction execution pipeline, with at most limited exceptions, is designed to process the RISC instructions independently

of whether the RISC instructions were decoded by the RISC instruction decoder or generated by the CISC instruction decoder.

26. (currently amended) The computer of claim 19, wherein the instruction execution pipeline, with at most limited exceptions, is designed to process the RISC instructions independently of a point within a recipe of a CISC instruction at which the RISC instruction was generated.

27. (original) The computer of claim 19, wherein the RISC and CISC instruction decoders are designed to emit RISC instructions to the instruction execution pipeline in a unified format with identical operational codings, differing at most by a source designator.

28. (original) The computer of claim 9, wherein the RISC instruction set has a condition-code based compare and branch repertoire.

29. (original) The computer of claim 9, wherein the RISC instruction set includes designators into a unified register file designed to contain integer and floating-point data, and the CISC instruction set includes designators into distinct integer and floating-point register files.

30. (original) The computer of claim 9, wherein intermediate results of multiple-side-effect instructions in the CISC instruction set are held in temporary registers of the computer that are not explicitly designated in the representations of the CISC instructions themselves; and

instructions of the RISC instruction set include designators into a register file, the RISC register designators including designators to the temporary registers used in the CISC instruction set.

31. (original) The computer of claim 9, further comprising a memory management unit designed to manage the instructions of the RISC and CISC instruction sets between a main memory of the computer and one or more cache levels.

32. (currently amended) A method, comprising the steps of:

executing a user-state program in a computer comprising a CISC hardware instruction decoder implementing less than an entire architectural definition of a CISC ~~an~~ instruction set, a remainder of the CISC instruction set being implemented in a software emulator, the software emulator being coded in a RISC instruction set for execution in a RISC hardware instruction decoder of the computer, the RISC instruction set of the RISC decoder designed to share a substantial number of opcode values with corresponding opcodes of the CISC instruction set, and implementing a substantial portion of the addressing modes of the CISC instruction set.

33. (original) The method of claim 32, wherein the computer comprises:

a RISC instruction decoder exposed for execution of user-state programs in a RISC instruction set, being an instruction set having fixed-length instructions and a load/store/operate organization; and

a CISC instruction decoder exposed for execution by user-state programs in a CISC instruction set, being an instruction set with variable-length instructions and many instructions having multiple side-effects;

an instruction execution pipeline designed to execute the output of both the RISC instruction decoder and the CISC instruction decoder.

34. (original) The method of claim 32, wherein the computer comprises a CISC instruction decoder and pipeline designed to decode and execute instructions of a complex instruction set having variable-length instructions and many instructions having multiple side-effects.

35. (original) The method of claim 34, wherein the computer further comprises processor register control circuitry designed to store information describing the decoding of the complex instructions into architecturally-visible processor registers of the computer.

36. (original) The method of claim 34, wherein the computer further comprises:

pipeline control circuitry designed to recognize an exception occurring in an instruction after a first side-effect of the instruction has been architecturally committed, to transfer control to a software exception handler of the emulator for the first exception, and to resume execution of the excepted instruction after completion of the exception handler, processor registers of the computer being designed to expose sufficient information about the state of the excepted instruction that the transfer and resume are effected without saving intermediate results of the excepted instruction on a memory stack.

37. (original) The method of claim 32, further comprising the steps of:

during execution of an instruction on the computer, in response to an operation of the instruction calling for an architecturally-visible side-effect in an architecturally-visible storage location, storing a value representative of an architecturally-visible representation of the side-effect, a format of the representative value being different than an architecturally-visible representation of the side-effect, and resuming the execution without generating the architecturally-visible side-effect;

later writing the architecturally-visible representation corresponding to the representative value into the architecturally-visible storage location.

38. (original) The method of claim 32, wherein some instructions of the program are executed entirely in the software emulator, and some instructions are partially implemented in the hardware instruction decoder and partially implemented in the software emulator.

39. (currently amended) A computer, comprising:

a hardware instruction decoder designed to decode a portion of a CISC ~~an~~ instruction set for the computer, and to deliver the decoded instructions to a multi-stage ~~instruction~~ execution pipeline ~~unit~~ for execution; and

a software emulator programmed to implement a remainder of the instruction set, the software emulator being coded in a RISC instruction set for execution in a RISC hardware instruction decoder of the computer, the RISC instruction set of the RISC decoder designed to share a substantial number of opcode values with corresponding opcodes of the CISC instruction set, and implementing a substantial portion of the addressing modes of the CISC instruction set.

40. (original) The computer of claim 39, further comprising:

a file of general registers, the instructions of the instruction set providing accessibility to only a subset of the general register file, intermediate results of instructions of the instruction set being stored in general registers of the general register file that are inaccessible in the instruction set.

41. (original) The computer of claim 40, the instruction set thereof being a first instruction set, and further comprising:

an exception handler coded in a second instruction set having accessibility to the general registers inaccessible in the first instruction set, the exception handler for initiation on an exception occurring at an intermediate point during execution of one of the instructions of the first instruction set, any saving of the intermediate results as part of a save of machine state using mechanisms used for saving general registers.

42. (original) The computer of claim 39, wherein:

the instruction decoder and an instruction unit are designed to decode and execute instructions of a complex instruction set having variable-length instructions and many instructions having multiple side-effects.

43. (original) The computer of claim 42, further comprising:

processor register control circuitry designed to store into architecturally-visible processor registers of the computer information describing the decoding of the complex instructions.

44. (original) The computer of claim 42, further comprising:

pipeline control circuitry designed to recognize an exception occurring in an instruction after a first side-effect of the instruction has been architecturally committed, to transfer control to a software exception handler for the first exception, and to resume execution of the excepted instruction after completion of the exception handler, processor registers of the computer being designed to expose sufficient information about the state of the excepted instruction that the transfer and resume are effected without saving intermediate results of the excepted instruction on a memory stack.

45. (original) The computer of claim 39:

wherein the execution unit comprises a multi-stage execution pipeline;

the instruction decoder being designed to generate information descriptive of instructions to be executed by the pipeline, to store the information into a non-pipelined register of the computer, to determine whether instructions will complete in the pipeline, and to abstain from writing descriptive information into the register for instructions following an instruction determined not to complete.

46. (original) The computer of claim 39, wherein:

at least some instructions of the instruction set have multiple side-effects and the potential to raise multiple exceptions; and

the pipeline exception circuitry is designed to recognize an exception occurring in an instruction after a first side-effect of the instruction has been architecturally committed, and

thereupon, to expose an instruction pointer, contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer exposing sufficient processor state and providing sufficient working storage for execution of the exception handler and resumption of the program, without recomputing the first side-effect, without storing processor state to the main memory.

47. (original) The computer of claim 39, further comprising:

circuitry designed to recognize an operation of an instruction calling for an architecturally-visible side-effect in an architecturally-visible storage location, and in response, to store a value representative of an architecturally-visible representation of the side-effect, a format of the representative value being different than an architecturally-visible representation of the side-effect, and to resume the execution without generating the architecturally-visible side-effect;

circuitry and/or software designed to later write the architecturally-visible representation corresponding to the representative value into the architecturally-visible storage location.

48. (original) The computer of claim 39, further comprising:

hardware designed to recognize a condition arising during execution of an instruction, in which the instruction is to affect the execution of a second instruction;

hardware and/or software designed to respond to the recognized condition by setting the computer into single-step mode; and

hardware and/or software designed to respond to execution of the second instruction by setting the computer out of single-step mode.

49. (original) The computer of claim 39, further comprising:

circuitry effective during execution of an instruction whose opcode calls for a memory reference to effect a movement of data, to perform a memory protection check

effective to check for permission to effect a movement of data other than the data movement called for by the instruction opcode.

50. (original) The computer of claim 39, further comprising:

circuitry designed to partially execute a post-termination loop iteration of a loop of a first instruction stream executing in the computer, after reaching a termination condition of the loop, the partial execution committing at least one side-effect to an architecturally-visible resource of the computer, and to raise an exception to transfer control to a second instruction stream;

software of the second instruction stream, programmed to unwind side-effects committed by the post-termination iteration.

51. (original) The computer of claim 39, further comprising:

instruction fetch and execution circuitry designed to fetch and execute instructions in two different instruction sets, each instruction set including store instructions to write data to a memory of the computer;

store monitoring circuitry designed to monitor the store instructions and to invalidate any copies of an old datum, including copies of instructions in an instruction cache in the instruction set other than the instruction set of the current store instruction.

52. (original) The computer of claim 39, wherein the emulator is coded in an instruction set other than the instruction set decoded by the instruction decoder.

53. (original) The computer of claim 39, wherein entry to the software emulator is by exception.

54. (original) The computer of claim 53, wherein entry to the software emulator is by exception inserted into the execution unit by the instruction decoder.

55. (original) The computer of claim 53, wherein the exceptions to enter the software emulator use the same pipeline and architectural infrastructure as other exceptions raised by the instruction decoder or instruction execution unit.

56. (original) The computer of claim 53, wherein some instructions are executed entirely in the software emulator, and some instructions are partially executed in the hardware instruction decoder and partially in the emulator.

57. (currently amended) A method, comprising the steps of:

executing a CISC portion of a program coded in a CISC ~~an~~ instruction set on a computer having a file of general registers, the CISC instruction set providing accessibility to only a subset of the general register file, intermediate results of instructions of the instruction set being stored in registers of the general register file that are inaccessible in the CISC instruction set; and

executing a RISC portion of the program coded in a RISC instruction set of the computer, the RISC instruction set designed to share a substantial number of opcode values with corresponding opcodes of the CISC instruction set and to implement a substantial portion of the addressing modes of the CISC instruction set.

58. (original) The method of claim 57, the instruction set thereof being a first instruction set, and further comprising the steps of:

servicing an exception occurring at an intermediate point during execution of one of the instructions of the first instruction set, the exception initiating an exception handler coded in a second instruction set having accessibility to the registers inaccessible in the first instruction set, any saving of the intermediate results as part of a save of machine state using mechanisms used for saving general registers.

59. (original) The method of claim 57, the instruction set thereof being a CISC instruction set with variable-length instructions and many instructions having multiple side-effects, and further comprising the steps of:

executing a user-state program coded in a RISC instruction set on the computer, the RISC instruction set being an instruction set having fixed-length instructions and a load/store/operate organization, the RISC instruction set providing accessibility to the entire general register file.

60. (original) The method of claim 57, wherein a hardware instruction decoder implements less than the entire instruction set, a remainder of the instruction set being implemented in a software emulator.

61. (currently amended) A computer, comprising:

a general register file of registers;

a CISC ~~an~~ instruction decoder designed to decode instructions of a CISC ~~an~~ instruction set of the computer, the instruction set providing accessibility to only a subset of the registers of the general register file, intermediate results of instructions of the instruction set being stored in registers of the general register file that are inaccessible in the instruction set; and

a RISC instruction decoder designed to decode a RISC instruction set of the computer, the RISC instruction set of the RISC decoder designed to share a substantial number of opcode values with corresponding opcodes of the CISC instruction set and to implement a substantial portion of the addressing modes of the CISC instruction set, and to have addressability to all registers of the general register file.

62. (original) The computer of claim 61, the instruction set thereof being a first instruction set, and further comprising:

an exception handler for initiation by an exception occurring at an intermediate point during execution of one of the instructions of the first instruction set, the exception handler being coded in a second instruction set having accessibility to the registers inaccessible in the first instruction set, any saving of the intermediate results as part of a save of machine state using mechanisms used for saving general registers.

63. (original) The computer of claim 61, wherein:

many individual instructions of the instruction set have multiple side-effects and the potential to raise multiple exceptions;

and further comprising pipeline exception circuitry designed to recognize an exception occurring in an instruction after a first side-effect of the instruction has been architecturally committed, and thereupon, to expose an instruction pointer, contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer exposing sufficient processor state and providing sufficient working storage for execution of the exception handler and resumption of the program, without recomputing the first side-effect, without storing processor state to the main memory.

64. (original) The computer of claim 61, further comprising:

circuitry designed to recognize an operation of an instruction calling for an architecturally-visible side-effect in an architecturally-visible storage location, and in response, to store a value representative of an architecturally-visible representation of the side-effect, a format of the representative value being different than an architecturally-visible representation of the side-effect, and to resume the execution without generating the architecturally-visible side-effect;

circuitry and/or software designed to later write the architecturally-visible representation corresponding to the representative value into the architecturally-visible storage location.

65. (original) The computer of claim 61, further comprising:

circuitry cooperatively designed with the instruction decoder and effective during execution of an instruction whose opcode calls for a memory reference to effect a movement of data, to perform a memory protection check effective to check for permission to effect a movement of data other than the data movement called for by the instruction opcode.

66. (original) The computer of claim 61, wherein:

the instruction decoder is designed, when decoding an instruction to write multiple operands to memory, to keep intermediate state of the instruction in the inaccessible registers.

67. (original) The computer of claim 61, wherein:

the instruction decoder is designed to store a single datum in parts in two or more of the registers.

68. (original) The computer of claim 67, wherein:

the instruction decoder is designed to generate instructions to store a single datum in parts in a plurality of the inaccessible registers, and to validate the single datum.

69. (original) The computer of claim 61, wherein:

the instruction decoder is designed to generate an instruction to compute a condition value into a one of the inaccessible registers during execution of a single instruction of the instruction set.

70. (original) The computer of claim 69, wherein:

the instruction decoder is further designed to generate an instruction to branch based on the condition value, and to leave the condition value dead before completion of the single instruction.

71. (original) The computer of claim 61, wherein:

the instruction decoder, general register file, and an instruction execution pipeline of the computer are cooperatively designed, such that execution of at least some single instructions results in computing multiple intermediate results being stored in a single inaccessible register.

72. (original) The computer of claim 61:

wherein, with few exceptions, all operations of the instructions in the instruction set that may generate exceptions are processed before any side effects of the instruction are committed to resources accessible in the first instruction set.